UFFS - UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CIÊNCIA DA COMPUTAÇÃO - 2020.2
CAMPUS CHAPECÓ

RELATÓRIO DO TRABALHO FINAL

Aluno: Artur Bernardo Mallmann

Maio 2021

RESUMO

Este trabalho visou a implementação de um compilador. Nele são usados os conhecimentos prévios adquiridos em matérias como LFA. As etapas abordadas nesta atividade de compilação são: Análise Léxica, Sintática, Semântica, geração de código intermediário, otimização

OBJETIVO

Criar uma linguagem definindo palavras chaves, a sintaxe, semântica e outras particularidades e então implementar os passos para a compilação das aplicações.

INTRODUÇÃO

Para a execução deste trabalho fiz uso das ferramentas re2c e GOLD Parser no apoio a análise léxica e semântica. O projeto infelizmente ficou inacabado, porém tem grande potencial. A linguagem se aproxima bastante do C, porém com diferenças como definir explicitamente funções com a palavra reservada “function” e não definir a tipagem das variáveis primitivas.

METODOLOGIA

Mesmo já tendo implementado o Autômato Finito Determinístico na Disciplina de LFA, decidi mudar de abordagem, então alternativamente troquei a implementação de Backus Naur Form que fiz anteriormente por uma que implementei com gramática regular e a ferramenta re2c.

Primeiramente definimos os token, reconhecimento de strings, números e outros símbolos:

```
/*!re2c
        re2c:define:YYCTYPE = char;
        re2c:yyfill:enable = 0;

        //erro:
        err = [0-9]+"\."?[0-9]*[_a-zA-Z];
//      null = [\s\t\n]*;
        return = "return";
        function = "function";
        loop = "loop";
        if = "if";
        else = "else";
        id = [a-zA-Z][_0-9a-zA-Z]*;
        num = [0-9]+"\."?[0-9]*;
        openc = "{";
..

        [\x00] { return 0; } // qualquer simbolo whitespace sai do loop
//      null {printf("_");return 0;}
        return {write_ts(saved,"RETURN", line, ts,0); goto loop;}
        function {write_ts(saved,"FUNCTION", line, ts,0); goto loop;}
        loop {write_ts(saved,"LOOP", line, ts,0); goto loop;}
        if {write_ts(saved,"IF", line, ts,0); goto loop;}
        else {write_ts(saved,"ELSE", line, ts,0); goto loop;}
        id {write_ts(saved,"ID", line, ts,0); goto loop;}
        num {
                write_ts(saved,"NUM", line, ts,atoi(saved));
                goto loop;}
        openc {write_ts(saved,"{", line, ts,0); goto loop;}
..
        err {write_ts(saved,"ERR", line, ts,0); return 0;}
        * {write_ts(saved,"ERR", line, ts,0); return 0;}
*/
```

Como podemos ver acima, além do contínuo reconhecimento da gramatical regular já fazemos a gravação das informações necessárias na tabela de símbolos. As saídas da etapa de Análise léxica são a Fita com os identificadores, erros e final de sentença e a tabela de símbolos.

A sintaxe definida para a linguagem é a seguinte:

```
"Case Sensitive" = 'True'
"Start Symbol" = <lib>

<ar> ::= '+' | '-' | OR
<ap> ::= '*' | '/' | '%' | AND
```

```
<cp> ::= '!='| '=='| '<' | '>'
<e> ::= <e> <cp> <r>
        | <r>
<r> ::= <r> <ar> <t>
        | <t>
<t> ::= <t> <ap> <f>
        | <f>
<f> ::= '(' <e> ')'
        | ID
        | NUM
        | STR
        | <call>

<cond> ::= IF <e> <stm> ELSE <stm>
<attr> ::= ID '=' <e>
<loop> ::= LOOP'('<e>')'<stm>
        |LOOP <stm>'('<e>')'
<call> ::= ID '('<param>
<com> ::= <loop>
        | <cond>
        | <attr>';'
        | <call>';'
        |<function>
<lib> ::= <function><lib>
        |<function>
        |'$'
<stm> ::= '{'<stm_>
        | <com>
        |RETURN <e>';'
<stm_> ::= <com><stm_> | '}'
        |RETURN <e> ';"}'
<function> ::= FUNCTION ID '('<param> <stm>
<param> ::= ID')' |')'
        | ID ',' <param>
```

Exemplo de código fonte aceito:

```
function nome(x){
        printf ( artur ) ;
        function aaa() { }
        a = 133;
        b = 133.22;
        aaa = "araturgads";
        return a + x;
}
function main() {return nome(x)};
```

CONCLUSÃO

Mesmo sendo apenas parcialmente concluído o projeto se mostrou promissor e com bom desempenho ao que tudo indica na análise léxica. A ferramenta GOLD Parser foi muito importante para o sucesso na construção da semântica.

REFERÊNCIAS BIBLIOGRÁFICAS

https://en.wikipedia.org/wiki/LALR_parser
http://re2c.org/manual/manual_c.html
APPEL, Andrew W. **Modern compiler implementation in C**. Cambridge university press, 2004.
Material da Disciplina